Report 0

Miguel Rodríguez, RA: 192744, Email: m.rodriguezs1990@gmail.com

I. Problem

The objective of this work is to perform some basic digital image processing

A. Mosaic

Build a mosaic of 4×4 blocks from a monochromatic image. The distribution of the blocks should be the numeration present in Fig. 1.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

(a) Image before processing

6	11	13	3
8	16	1	9
12	14	2	7
4	15	10	5

(b) Image after processing

Fig. 1: Mosaic distribution

B. Intensity transformation

Perform two intensity transformation of a monochromatic image:

1) Get negative of a image as in Fig. 2.
2) Convert intensity interval to [100, 200].

C. Combination of images

Combine two monochrome images of the same size by weighted average of their gray levels, as shown in Fig. 3.

II. Solution

In this section we will describe the different solutions that were given to the problems raised in section I.

A. Solution to the mosaic problem

In order to solve the mosaic problem it's necessary decompose it into two different sub-problems: split image I into $N \times N$ blocks and create new a image from a random order of blocks.

It was decided to generalize the problem, the value N of blocks will be entered into the system by user, also the blocks will be ordered randomly in final image, which will generate a different final image in each instance of the program.

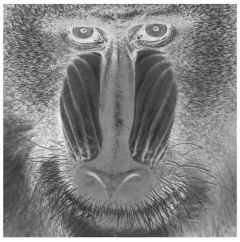
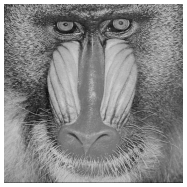


Fig. 2: Negative Image

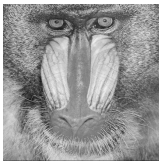


(a) Image A

(b) Image B

(c) 0.2*A+0.8*B

(d) 0.5*A+0.5*B



(e) 0.8*A+0.2*B

Fig. 3: Images before and after combination

1) Split image into $N \times N$ blocks: To divide an image I with width w and height h into $N \times N$ blocks it's necessary that $w \ mod(N) = 0$ and $h \ mod(N) = 0$, if this is not true, image I must be preprocessed.

The preprocessing consists of obtaining a new image of size $w' \times h'$, where $w' \ leq \ w \ \wedge w' \ mod(N) = 0$ and $h' \leq h \wedge h' \ mod(N) = 0$. The $preprocessing(image, n\_tiles)$ function described in Listing 1 was created, which allows this treatment to be performed on the input image.

```
def preprocessing(img, n_tiles):
 height, width = img.shape

 new_width = (width // n_tiles) * n_tiles
 new_height = (height // n_tiles) * n_tiles

 new_img = img[0:new_height, 0:new_width]

 return new_img
```

Listing 1: Image preprocessing

After performing the preprocessing of image, we proceed to divide image into $N \times N$ frames. This division of I was made using the notation of slices provided by python, which has as input the initial and final range for each dimension of the matrix, these ranges are described in the equations (1), (2), (3) y (4).

$$row\_initial\_range = floor\left(\frac{i*h}{N}\right) \quad (1)$$

$$row\_end\_range = floor\left(\frac{i*h}{N}\right) - 1 \quad (2)$$

$$col\_initial\_range = floor\left(\frac{i*w}{N}\right) \quad (3)$$

$$col\_end\_range = floor\left(\frac{i*w}{N}\right) - 1 \quad (4)$$

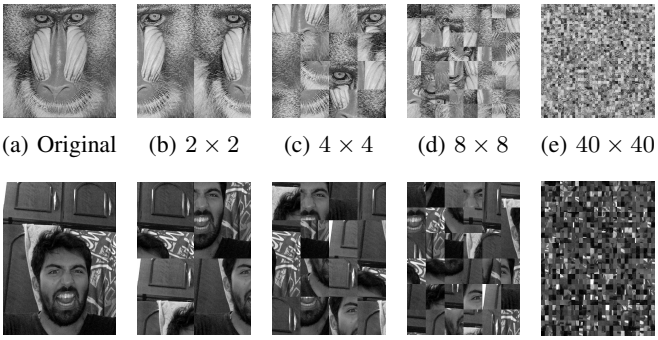
Where i and j are loop indexes. The loops iterate $N \times N$ times creating $N \times N$ blocks of split image I. This can be seen in function $tiled(image, N)$ in the Listing 2. This function returns an list of $N \times N$ matrices, which represent split image I.

```
def tiled(img, n_tiles):
 tiles = []
 height, width = img.shape

 for i in range(0,n_tiles):
  for j in range(0,n_tiles):
   row_range_lower = (i * height)//n_tiles
   row_range_upper = (((i + 1) * height)//n_tiles) -
     1

   col_range_lower = (j * width)//n_tiles
   col_range_upper = (((j + 1) * width)//n_tiles) -
    1

   tiles.append(img[row_range_lower:row_range_upper,
    col_range_lower:col_range_upper])

 return tiles
```

Listing 2: Image split function

2) Create new a image from a random order of blocks: After obtaining a list of $N \times N$ matrices created from image i, it's necessary create the new image with a new order as shown in Fig. 1b. We decided to generalize the creation of image I', using a random way of mixing matrices, for this we created a list of indexes of size $N \times N$, wich are disordered using the function $random.shiffle()$ provided by python random library. With this new index order we begin to construct a new image I' as shown in Listing 3. For case when the number of blocks is 4×4, we use order established in Fig. 1b.

After performing different tests with code in Listing 7, we obtained the results shown in Fig. 4, where we see processing of two images 4a and 4f with different sizes of $N \times N$. First processing was performed with square image shown in Fig. 4a. This was processed with different block sizes $N \times N$. The results are shown in Figs. 4b, 4c, 4d and 4e, we can observe that there is no problem when processing different sizes of blocks, as long as they are smaller than the amount



(a) Original (b) 2×2 (c) 4×4 (d) 8×8 (e) 40×40

(f) Original (g) 2×2 (h) 4×4 (i) 8×8 (j) 40×40

Fig. 4: Images resulting from processing with differents numbers of blocks.

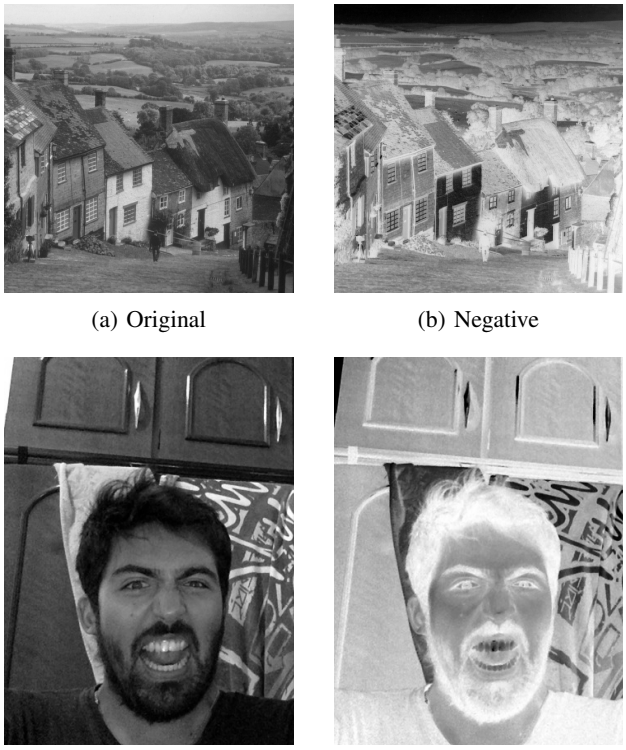
of pixels that are of width and height. The second processing was performed using different block sizes $N \times N$ to Fig. 4f. The results can be seen in Figs. 4g, 4h, 4i and 4j, with these results we can conclude that the algorithm work for non-square images, so this program works for all types of images supported by the scipy python library. The only consideration is that the height and width of the image I should be greater than the number of blocks $N \times N$

```
def create_mosaic(img, n_tiles=4):
 img = preprocecing(img, n_tiles)
 tiles = tiled(img, n_tiles)

 mosaic_order = np.arange(n_tiles**2)
 random.shuffle(mosaic_order)
 mosaic_order = mosaic_order.reshape((n_tiles,
     n_tiles)) if n_tiles != 4 else np.array([5, 10,
     12, 2, 7, 15, 0, 8, 11, 13, 1, 6, 3, 14, 9, 4]).
     reshape((n_tiles, n_tiles))

 for i in range(0,n_tiles):
  for j in range(0, n_tiles):
   if j == 0:
    new_row = tiles[mosaic_order[i][j]]
   else:
    new_row = np.concatenate((new_row,tiles[
     mosaic_order[i][j]]), axis=1)
  if i == 0:
   new_img = new_row
  else:
   new_img = np.concatenate((new_img, new_row), axis
    =0)

 return new_img
```

Listing 3: Create mosaic function

B. Solution to the intensity transformation problem

1) Get negative of a image: It is necessary to understand two basic concepts before you can perform the negative of an image: First, the range of possible values of the pixels of an image I can vary depending on the representation of this, for the case of our solution, all the images are represented with a range of possible values between 0 and 255. Second, it is



(a) Original (b) Negative

(c) Original (d) Negative

Fig. 5: Original and negative images.

necessary to understand that in order to obtain the negative of an image, a transformation of the values of $pixels$ of image I must be made, this transformation consists in changing the value of $pixel$ with its opposite in the scale, ie if the original $pixel$ is 0, it must be transformed to 255, if 1 becomes 254, so we can deduce eq. (5). Wich was implemented in the Listing 4

$$I'(i,j) = 255 - I(i,j) \tag{5}$$

The Results of applying function in Listing 4 on different images can be seen in Fig. 5, in which Figs. 5a and 5c are original images and Figs. 5b and 5d are resulting images from apply the eq. (5).

```
def negative(image):
 return 255 - image
```

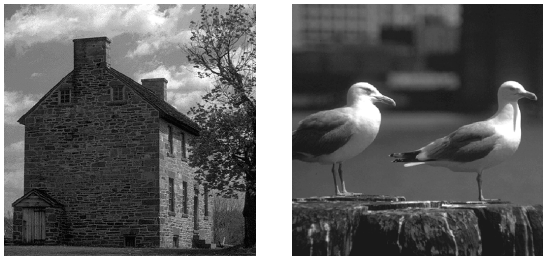
Listing 4: Negative function

2) Convert intensity interval to [100, 200]: We decided to generalize this problem, so that the new interval would be $[n, m]$ where $0 \leq n \leq m \leq 255$, which allows us to perform any type of linear transformation on intensity of Image I, while it is defined with a range of $[0, 255]$. This transformation is described in eq. (6) and implemented in Listing 5.

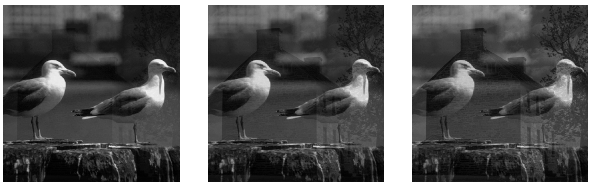
$$I'(i,j) = \left(\frac{m-n}{255}\right) * I(i,j) + n \tag{6}$$

(a) Original (b) [0 - 150] (c) [100 - 200] (d) [150 - 255]

Fig. 6: Images resulting from processing with differents ranges of transformation.



(a) Image A (b) Image B



(c) $0.1 \cdot A + 0.9 \cdot B$ (d) $0.2 \cdot A + 0.8 \cdot B$ (e) $0.3 \cdot A + 0.7 \cdot B$

(f) $0.5 \cdot A + 0.5 \cdot B$ (g) $0.6 \cdot A + 0.4 \cdot B$ (h) $0.8 \cdot A + 0.2 \cdot B$

Fig. 7: Images resulting from combination of A and B with different multiplicative factors.

In Fig. 6 we can see the results of applying eq. (6), where original image is Fig. 6a and Figs 6b, 6c, 6d are resulting images from applying transformations with ranges of $[0, 150]$, $[100, 200]$ and $[150, 255]$ respectively. The complete code of this section can be seen in the Listing 8. As in previous problem, this code can be used with any kind of images supported by scipy python library.

```
def transformation(image, initial_range, final_range
    ):
 return np.uint8(((final_range - initial_range)*
    image)//255 + initial_range)
```

Listing 5: Transformation function

C. Combination of images

As in the other problems, it was decided to generalize the solution. For this, a program was created that would allow the combination of two images I_1 and I_2, using different multiplicative factors. The method used to perform this combination can be seen in the eq. (7) or in its implementation in Listing 6.

$$I'(i,j) = p_1 * I_1(i,j) + p_2 * I_2(i,j) \tag{7}$$

```
def merge(img1, img2, percentage1, percentage2):
 return percentage1*img1 + percentage2*img2
```

Listing 6: Combination function

Different experiments were performed by mixing two images (Fig. 7a and 7b), using different multiplicative factors. In Figs. 7c and 7d we use a factor favoring Fig. 7b, where can see that this is the predominant in the combination. In the Figs. 7g and 7h the multiplicative factor favors Fig. 7a. Can also observe that using a factor that similar weights on two images can give results where both image have similar weights at the level of visual information, as in Fig. 7f and 7g. The code used to solve this problem can be seen in Listing. 9.

APPENDIX A
MOSAIC CODE

```
import sys
import random

from scipy import misc
import numpy as np

def tiled(img, n_tiles):
 tiles = []
 height, width = img.shape

 for i in range(0,n_tiles):
  for j in range(0,n_tiles):
   row_range_lower = (i * height)//n_tiles
   row_range_upper = (((i + 1) * height)//n_tiles) -
     1

   col_range_lower = (j * width)//n_tiles
   col_range_upper = (((j + 1) * width)//n_tiles) -
    1

   tiles.append(img[row_range_lower:row_range_upper,
    col_range_lower:col_range_upper])

 return tiles

def preprocessing(img, n_tiles):
 height, width = img.shape

 new_width = (width // n_tiles) * n_tiles
 new_height = (height // n_tiles) * n_tiles

 new_img = img[0:new_height, 0:new_width]

 return new_img


def create_mosaic(img, n_tiles=4):
 img = preprocessing(img, n_tiles)
 tiles = tiled(img, n_tiles)

 mosaic_order = np.arange(n_tiles**2)
 random.shuffle(mosaic_order)
 mosaic_order = mosaic_order.reshape((n_tiles,
    n_tiles)) if n_tiles != 4 else np.array([5, 10,
    12, 2, 7, 15, 0, 8, 11, 13, 1, 6, 3, 14, 9, 4]).
    reshape((n_tiles, n_tiles))

 for i in range(0,n_tiles):
  for j in range(0, n_tiles):
   if j == 0:
    new_row = tiles[mosaic_order[i][j]]
   else:
    new_row = np.concatenate((new_row,tiles[
    mosaic_order[i][j]]), axis=1)
  if i == 0:
   new_img = new_row
  else:
   new_img = np.concatenate((new_img, new_row), axis
    =0)

 return new_img


def main(argv):
 if (len(argv) < 4):
  print("Incorrect number of arguments")
  print("Execute: python3 filename.py filename_in
    filename_out number_of_tiles")
  return -1

 filename_in = argv[1]
 filename_out = argv[2]
 n_tiles = int(argv[3])

 image = misc.imread(filename_in, True)
 new_img = create_mosaic(image, n_tiles)

 misc.imsave(filename_out, new_img)


if __name__ == "__main__":
 sys.exit(main(sys.argv))
```

Listing 7: Mosaic code

APPENDIX B
TRANSFORMATION CODE

```
import sys

from scipy import misc
import numpy as np

def negative(image):
 return 255 - image

def transformation(image, initial_range, final_range
    ):
    return np.uint8(((final_range - initial_range)*
    image)//255 + initial_range)

def main(argv):
 if (len(argv) < 6):
  print("Incorrect number of arguments")
  print("Execute: python3 filename.py filename_in
    filename_negative filename_transformation
    initial_range final_range")
  return -1

 filename_in = argv[1]
```

```
 filename_negative = argv[2]
 filename_transformation = argv[3]
 initial_range = int(argv[4])
 final_range = int(argv[5])

 image = misc.imread(filename_in, True)

 img_negative = negative(image)
 img_transformation = transformation(image,
    initial_range, final_range)

 misc.imsave(filename_negative, img_negative)
 misc.imsave(filename_transformation,
    img_transformation)

 return 0

if __name__ == "__main__":
 sys.exit(main(sys.argv))
```

Listing 8: Transformation code

APPENDIX C
COMBINATION CODE

```
import sys

from scipy import misc
import numpy as np


def merge(img1, img2, percentage1, percentage2):
 return percentage1*img1 + percentage2*img2

def main(argv):
 if (len(argv) < 6):
  print("Incorrect number of arguments")
  print("Execute: python3 file1 file2 percentage1
    percentage2 filename_out")
  return -1

 filename1 = argv[1]
 filename2 = argv[2]
 percentage1 = float(argv[3])
 percentage2 = float(argv[4])
 filename_out = argv[5]

 if percentage1 > 1.0 or percentage2 > 1.0 or
    percentage1 < 0.0 or percentage2 < 0.0:
  print("Percentage must be greater than 0.0 and
    less than 1.0")
  return -1

 if percentage1 + percentage2 != 1.0:
  print("Percentage1 plus percentaje2 must be 1.0")
  return -1


 image1 = misc.imread(filename1, True)
 image2 = misc.imread(filename2, True)

 if image1.shape != image2.shape:
  print("Images must be the same size")
  return -1

 merged_image = merge(image1, image2, percentage1,
    percentage2)

 misc.imsave(filename_out, merged_image)

 return 0

if __name__ == "__main__":
 sys.exit(main(sys.argv))
```

Listing 9: Combination code

Miguel Rodriguez IEEE Member: 93224316, RA: 192744. MSc Student from UNICAMP, Brazil. Computers and Telecommunications Engineer from Diego Portales University, Chile. A charismatic young boy, empathic, a history and culture lover, motivated to travel around the world and learn about different cultures we can find in our planet.

The most noticed abilities are: the capacity to work as a team with people from different areas, the power to surpass problems without losing the desire and motivation to solve them, the great capacity and encouragement to learn new and interesting staffs, being this one, the best ability I earned during my college period.

Very interested in keep improving the skills in labour and the academic field; always searching new technologies and new techniques and moral challenges, especially in areas like artificial intelligence. A bike lover and technologies which use renewable energy, this due to the big motivation to build a future where technology and science can pacifically coexist with nature and that way contribute to improve the life quality in society.